

Distributed consensus with mixed time/communication bandwidth performance metrics

Federico Rossi

Marco Pavone

Abstract—In this paper we study the inherent trade-off between time and communication complexity for the distributed consensus problem. In our model, communication complexity is measured as the maximum data throughput (in bits per second) sent through the network at a given instant. Such a notion of communication complexity, referred to as bandwidth complexity, is related to the frequency bandwidth a designer should collectively allocate to the agents if they were to communicate via a wireless channel, which represents an important constraint for dense robotic networks. We prove a lower bound on the bandwidth complexity of the consensus problem and provide a consensus algorithm that is bandwidth-optimal for a wide class of consensus functions. We then propose a distributed algorithm that can trade communication complexity versus time complexity as a function of a tunable parameter, which can be adjusted by a system designer as a function of the properties of the wireless communication channel. We rigorously characterize the tunable algorithm’s worst-case bandwidth complexity and show that it compares favorably with the bandwidth complexity of well-known consensus algorithm.

I. INTRODUCTION

Distributed decision-making in robotic networks is a ubiquitous problem, with applications as diverse as state estimation [1], formation control [2], tracking [3] and cooperative task allocation [4]. Distributed decision-making problems such as leader election, majority voting, distributed hypothesis testing, and some distributed optimization problems can all be abstracted as instances of the *consensus* problem, where nodes in a robotic network have to agree on some common value [5], [6]. For this reason, the consensus problem has gathered significant interest in the Control Systems community in recent years, following the seminal works in [7], [8], [9].

Research in the Control Systems community has mainly focused on the *average* consensus problem and, in particular, on local averaging algorithms, where nodes repeatedly average their state with their neighbors’: fundamental limitations on the *time* complexity of local averaging algorithms are now known [10]. Conversely, research in the Computer Science community has mainly focused on lower bounds on the general consensus problem in presence of node failures (both non-malicious and byzantine): several lower bounds on the time and communication complexity of the general consensus problem are now known. Significant attention has also been devoted to the *leader election* problem, a specific consensus problem where agents in a network select a single agent as their leader. Fundamental limitations of the

leader election problem in terms of time and communication complexity and matching optimal algorithms are now well-understood [6]. However, complexity results in the Control Systems and in the Computer Science communities strongly rely on the assumption that all messages can be delivered within a *finite* amount of time that does *not* depend on message size [6]. This assumption has significant effects on the *frequency bandwidth* collectively required by the agents: however, despite the large interest in the consensus problem and its applications, the problem of bandwidth use has seen very limited investigation in both the Computer Science and the Control Systems communities.

Motivation: In this paper we argue that bandwidth use can play a significant role in the real-world performance of consensus algorithms and significantly limit their scalability as the number of agents increases.

In order to appreciate the importance of bandwidth use in modern robotic networks, consider the following scenario. A network of $n = 100$ robotic agents, each with six mechanical degrees of freedom is tasked with averaging their state (i.e. performing average consensus), e.g. to control their formation [2] or to filter noisy observations of a common target [1]. Each agent’s state or observation can be represented by twelve floating point numbers: the size of each agent’s initial condition is $b = 768$ bits. Each message should be delivered in 10 ms at most, in order to guarantee acceptable time performance. The complexity results in Section IV allow us to show that a bandwidth-optimal algorithm such as GHS with convergecast achieves consensus with a bandwidth complexity of 143 kbps (and converges in approximately 6.7 s). The popular average-based consensus algorithm requires 7750 kbps and has a significantly slower convergence rate, achieving convergence in approximately 100 s. Finally, the time-optimal flooding algorithm only requires 1 s to converge, but it requires a bandwidth of 775 Mbps. As a comparison, a bandwidth-optimized protocol such as 802.11n WiFi requires use of the entire 2.4 GHz ISM band to transmit 288Mbps over very short distances: thus, selecting an inappropriate consensus algorithm can have a dramatic impact on the real-world performance, the scalability and potentially even the stability of a cyber-physical network.

Statement of contributions: The contribution of this paper is threefold. First, we propose a rigorous metric for the bandwidth complexity of decentralized algorithms with omnidirectional (broadcast) communication channels and discuss its relevance to modern media access control (MAC) mechanisms. Second, we prove a lower bound on the band-

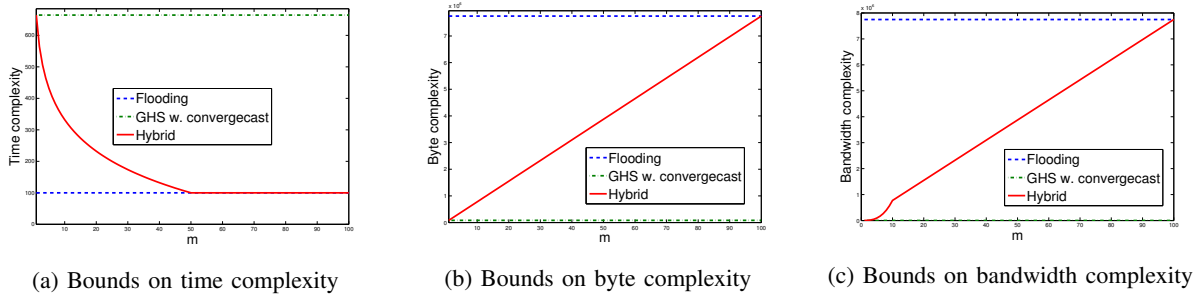


Fig. 1: Bounds on time, byte and bandwidth complexity of the flooding algorithm (in blue, dashed), GHS algorithm with convergecast (in green, dash-dotted) and of our hybrid algorithm (in red, solid) as a function of the tuning parameter m for a network of size $n = 100$ with state size $b = 768$ bits. Our hybrid algorithm recovers the time-optimal performance of the flooding algorithm as $m \rightarrow n$. Conversely, the hybrid algorithm recovers the byte and bandwidth performance of the byte-optimal and bandwidth-optimal GHS algorithm with convergecast, presented in Section IV, as $m \rightarrow 1$.

width complexity of the generalized consensus problem. The bound is tight for a wide class of consensus functions that includes many distributed consensus problems such as mean and weighed mean, leader election, selected distributed optimization problems and majority voting. Finally, we show that the hybrid algorithm proposed by the authors in [11] achieves intermediate bandwidth performance between the time-optimal flooding algorithm and the bandwidth-optimal GHS algorithm with convergecast. This result, combined with our previous findings, shows that our hybrid algorithm can be tuned to satisfy mixed time-bandwidth metrics as well as mixed time-energy metrics, trading time complexity (and, to an extent, robustness) for spectrum utilization and energy consumption. A graphical depiction of our lower bounds on time, byte and bandwidth complexity of the flooding algorithm, the GHS algorithm with convergecast and our hybrid algorithm is shown in Figure 1.

Organization: This paper is organized as follows. In Section II, after formalizing our agent model and network model, we give a formal definition of bandwidth complexity and justify its relevance to modern multi-agent media access control mechanisms. We then provide a rigorous definition of the generalized consensus problem. In Section III we prove a lower bound on the bandwidth complexity of the generalized consensus problem. In Section IV we prove tightness of the lower bound. We also study the bandwidth complexity of common consensus algorithms (including the flooding algorithm, the GHS algorithm with convergecast and the average-based consensus-algorithm). A variation of the GHS algorithm with convergecast achieves the lower bound presented in the previous section. In Section V, we study the bandwidth complexity of the hybrid algorithm introduced by the authors in [11]. We show that the bandwidth complexity of the algorithm smoothly transitions from bandwidth-optimal performance to the performance of the time-optimal flooding algorithm. Finally, in Section VI, we draw our conclusions and discuss directions for future research.

II. PROBLEM SETUP

In this section we introduce the model and the complexity metrics used in this work. We also describe naming conventions that will be used in the rest of this paper. A preliminary version of the model has appeared in [11].

A. Agent model

An agent in a robotic network is modeled as an input/output (I/O) automaton, i.e., a labeled state transition system able to send messages, react to received messages and perform arbitrary internal transitions based on the current state and messages received. A precise definition of I/O automaton is provided in [6, pp. 200-204] and is omitted here in the interest of brevity. All agents in a robotic network are identical except for a unique identifier (UID - for example, an integer). The time evolution of each agent is characterized by two key assumptions:

- **Fairness assumption:** the order in which transitions happen and messages are delivered is not fixed a priori. However, any enabled transition will *eventually* happen and any sent message will *eventually* be delivered.
- **Non-blocking assumption:** every transition is activated within l time units of being enabled and every message is delivered within d time units of being dispatched.

Essentially, the fairness assumption states that each agent will eventually have an opportunity to perform transitions, while the non-blocking assumption gives timing guarantees (but no synchronization). We refer the interested reader to [6, pages 212-215] for a detailed discussion of these assumptions. We argue here that these are *minimal* assumptions for most real-world robotic networks.

B. Network model

A robotic network comprising n agents is modeled as a *connected, undirected* graph $G = (V, E)$, where $V = \{1, \dots, n\}$ is the node set, and $E \subset V \times V$, the edge set, is a set of *unordered* node pairs modeling the availability of a communication channel. Two nodes i and j are neighbors if $(i, j) \in E$. The neighborhood set of node $i \in V$ is the set of nodes $j \in V$ that are neighbors of node i . Henceforth, we

will refer to nodes and agents interchangeably. Our model is *asynchronous*, i.e., computation steps within each node and communication are, in general, asynchronous. In this paper we focus on *static networks*, i.e., robotic networks where the edge set does not change during the execution of an algorithm. However, we do remark that (i) our lower bounds also apply to time-varying networks (although, of course, they may not be tight) and (ii) the hybrid algorithm described in section V has *limited, tunable* resistance to network disruption, requiring a tunable amount of time to reconfigure after network disruptions. We refer the interested reader to [11] for a thorough discussion of the recovery mechanism and of its time complexity.

C. Model of communication

Nodes communicate with their neighbors according to a *local broadcast* communication scheme. A node can send a message to all neighbors simultaneously: the cost of a message (in terms of energy consumption and bandwidth use) is independent of the number of receivers. We remark that local broadcast algorithms can emulate one-to-one communication: it is sufficient to append the intended recipient's UID to each broadcast and instruct non-recipients to ignore the message.

The local broadcast communication scheme is representative of robotic networks where agents are equipped with *omnidirectional* antennas: this arrangement is typical of most current airborne and ground-based robotic networks, where steerable antennas are unadvisable due to the agents' mobility.

Message transmission requires a finite, nonvanishing amount of time; the non-blocking assumption ensures that every message is delivered within time d . We consider d to be constant: that is, all messages are delivered within a maximum time that does *not* depend on message size or type. This assumption is widely used in the Computer Science community [6] and is typical of TCP-like communication protocols. Thus, the parameter d represents a *desired performance level*. *Collisions* occur when (part of) two or more messages are transmitted on the same frequency at the same time: when a collision occurs, all messages involved in the collision are not delivered. When analyzing bandwidth complexity, we assume that messages are sent at a *constant rate* throughout a window of length d : it is easy to see that "bursty" transmission would only decrease bandwidth performance.

We remark that, in presence of a TCP communication protocol, collisions do not cause messages to be permanently lost: nodes can sense the collision and resend the information at a later time. However, frequent collisions can have a major impact on the time required to deliver a message and, as a result, on the execution time of an algorithm.

We also remark that, in *directional* communication schemes, communication channels can be *spatially* separated: that is, two messages may be transmitted on the same frequency at the same time with minor interference if the respective directional communication channels do not

physically overlap. In our omnidirectional communication scheme, on the other hand, collisions occur whenever two nodes within range of one another send messages on the same frequency at the same time; in addition, even if two nodes are not within range of one another, collisions may occur if they are trying to contact a third node in range of both (this problem is known as the "hidden node problem" in the telecommunication community). In this paper we assume no restrictions to the network topology the nodes can assume: we remark that there exist both dense and sparse network topologies where every node's communications may spatially interfere with all other nodes'.

D. Bandwidth complexity measure

We define bandwidth complexity as the infimum worst-case (over graph topologies, initial values, fair executions and execution time) overall number of bytes transmitted at the same instant by all agents in the network.

Let \mathcal{G} be a set of graphs with node set $V = \{1, \dots, n\}$. For a given graph $G \in \mathcal{G}$, let $\mathcal{F}(a, x, G)$ be the set of *fair executions* for an algorithm $a \in \mathcal{A}$ and a set of initial conditions $x \in \mathcal{X}^n$ (a fair execution is an execution of an algorithm that satisfies the fairness and non-blocking assumptions stated above).

Rigorously, the bandwidth complexity for a given consensus function f with respect to the class of graphs \mathcal{G} is

$$\text{FC}(f, \mathcal{G}) := \inf_{a \in \mathcal{A}} \sup_{G \in \mathcal{G}} \sup_{x \in \mathcal{X}^n} \sup_{\alpha \in \mathcal{F}(a, x, G)} \sup_{t \in [0, T(a, x, \alpha, G)]} F(a, x, \alpha, G, t)$$

where $F(a, x, \alpha, G, t)$ is the bandwidth (measured by the size of all messages transmitted at time t divided by the maximum transmission time d) at time t of execution α of algorithm a with initial conditions x on a graph G . While very simple, the bandwidth complexity measure is a reliable proxy for many wireless communication protocols and media access control (MAC) mechanisms. Its interpretation varies depending on the specific MAC mechanism employed:

- If Frequency Division Multiple Access (FDMA) is employed, the frequency bandwidth required by a single message is proportional to the message size divided by the maximum transmission time d ; the overall frequency bandwidth required by the network is proportional to the maximum *overall* size of messages being transmitted at a given time divided by the maximum transmission time d , since every message must be broadcast on a different frequency slot;
- If a Time Division Multiple Access (TDMA) media access control mechanism is employed, each agent is allocated a time slot in a round robin fashion so that only one agent can transmit during a given time slot. The sum of the durations of all time slots must be smaller than d to guarantee that all sent messages be delivered within d time units: thus, in order to guarantee a timely delivery, the frequency bandwidth required must be proportional to the maximum overall size of all messages sent at any instant of time, divided by d .

- If Code Division Multiple Access (CDMA) is employed, multiple messages are relayed on the same, wide frequency spectrum at the same time; a *spread spectrum* technique is employed to make decoding of sent messages possible. The bandwidth of the spread spectrum is significantly larger than the bandwidth of the uncoded signal: in particular (i) the bandwidth required by a single message before encoding is proportional to its size (in bytes) and (ii) the spreading gain is roughly proportional to the maximum number of users that the network can support. Thus, if all agents transmit messages of the same size at the same time (as will be the case for the proof of the lower bound we study in this paper and also for the algorithms we discuss in Section IV), bandwidth complexity captures the frequency spectrum required for successful communication with a CDMA MAC mechanism.
- A rigorous study of the effect of available bandwidth on channel capacity when collision-detection mechanisms such as CSMA/CA are employed is beyond the scope of our work. However we remark that, for a given message size, increasing bandwidth reduces the time required to transmit a message and thus the network load, significantly reducing the probability of collisions and thus increasing the effective throughput.

Furthermore, regardless of the MAC mechanism employed, for large signal-to-noise ratios, the maximum capacity of a wireless channel is approximately proportional to bandwidth, as shown by Shannon in [12].

E. Model of computation

In this paper we study collective decision-making problems where each node in the network is endowed with an initial value x_i (which can be represented with b bits) and *each* node should output the value of a function of the initial values of *all* nodes. In other words, each agent, after exchanging messages with its neighbors and performing internal state transitions, should output $f(x_1, \dots, x_n)$ for some computable function f , which we call a *consensus function*. We formalize the notions of consensus function as follows.

Consensus functions: A consensus function is a *computable* function $f : \mathcal{X}^n \mapsto \mathbb{R}$ that depends on *all* its arguments. More precisely, for each element $x = (x_1, \dots, x_n) \in \mathcal{X}^n$ and for all $i \in \{1, \dots, n\}$ one can find elements $x_i^{(1)} \in \mathcal{X}$ and $x_i^{(2)} \in \mathcal{X}$ such that

$$f(x_1, \dots, x_i^{(1)}, \dots, x_n) \neq f(x_1, \dots, x_i^{(2)}, \dots, x_n).$$

Loosely speaking, such choice of consensus function implies that each node is needed for the decision-making process. We collectively refer to problems involving the distributed computation of consensus functions (as defined above) as *generalized consensus*.

We introduce a *representation* property for consensus functions that will be instrumental to derive fundamental limitations of performance in terms of the amount of information exchanged.

Hierarchically computable consensus function: A consensus function is hierarchically computable if it can be written as the composition of a commutative and associative binary operator $*$, that is

$$f(x_1, x_2, \dots, x_n) = x_1 * x_2 * \dots * x_n.$$

(The name is inspired by the observation that hierarchically computable functions can be computed with messages of small size on a *hierarchical* structure such as a tree). Furthermore, for a consensus function to be hierarchically computable, the consensus value as well as all intermediate products should be of the same size (in bytes) as the initial value. That is, if storing x_i requires $\Theta(b)$ bytes, then storing the result of the operation $(x_i * x_j)$ and of the consensus value $f(x_1, \dots, x_n)$ should also require $\Theta(b)$ bytes.

We remark that the class of hierarchically computable consensus functions includes average and weighed average, MAX and MIN (often used in leader election), voting and selected distributed optimization problems. We refer the reader to [13] for a more exhaustive characterization of this class of functions.

F. Nomenclature

In the rest of this paper, we use the following definitions for nodes belonging to a rooted tree structure.

- Each rooted tree contains a *root node*.
- A node j is the *child* of node i if (i) the shortest path from node j to the root is one edge longer than the shortest path from node i to the root and (ii) node i and node j share an edge in the tree. Conversely, node i is called node j 's *parent*.
- A *leaf node* is a node with no children.
- A node j is a *descendant* of node i if (i) the shortest path from node j to the root is strictly longer than the distance from node i to the root and (ii) the path connecting node j and the root node contains node i .
- The set containing a node j and all its descendants is the *branch* of node j .

Figure 2 shows a graphical depiction of the definitions above.

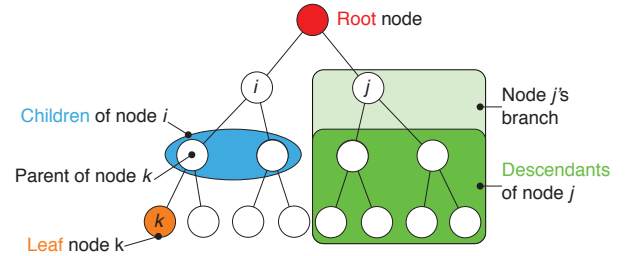


Fig. 2: Naming conventions for nodes belonging to a rooted tree structure

III. LOWER BOUNDS ON BANDWIDTH COMPLEXITY OF CONSENSUS

In this section, we present two lower bounds on the bandwidth complexity of the consensus problem. The first

bound applies to *asynchronous* executions and only depends on very mild assumptions on the minimum number of available UIDs; the second bound applies to synchronous as well as asynchronous executions and requires slightly more restrictive assumptions on the execution time and on the minimum number of available UIDs.

Proposition III.1 (Lower bound on the bandwidth complexity of the consensus problem in asynchronous executions). *Assume that messages carry a sender and/or a receiver ID. Assume also that agents' UIDs are selected from a set S of cardinality $|S| \geq 2n$. Then, for a given consensus function f and class of graphs \mathcal{G} with n nodes, $FC(f, \mathcal{G}) \in \Omega((n \log n + b)/d)$.*

Proof. When they start execution, agents decide whether to send a message or whether to wait in silence until they hear a message. Since agents have no information about other nodes before they receive a message, their decision is based solely only on their UID and, possibly, their initial condition. We consider asynchronous executions: agents do not have access to a shared clock and therefore can not make decisions based on time.

We wish to show that, if the agents' UIDs are drawn from a set U with cardinality $|U| \geq 2n$, then there exist a subset $M \subseteq U$ with cardinality $|M| \geq n$ such that all agents with an UID from M send a message before receiving a message at least for one set of initial conditions.

We proceed by contradiction. Call $S = U \setminus M$ the set of UIDs such that, for all initial conditions, an agent sends no message before receiving one message. Now, assume by contradiction that the cardinality of M is smaller than n . Then the cardinality of S is larger than n : there exists an assignment of n UIDs to the agents such that no node in the network sends a message before receiving one. Any network formed from these agents exchanges no messages and, in particular, it fails to solve the consensus problem unless all initial conditions are identical. We have reached a contradiction.

If the cardinality of M is larger than n , then an adversary can select n UIDs and n matching initial conditions from M and form a network where every node sends a message before receiving one. In an asynchronous executions, all nodes can transmit their first message simultaneously. Furthermore, at least $\log n$ bits are required to store n distinct UIDs: thus, if messages contains the transmitter or the receiver UID, the size of each message is lower-bounded by $\log n$. Therefore $n \log n$ bits may be transmitted simultaneously, with a bandwidth complexity of $n \log n/d$.

Finally, we observe that every agent transmits its initial value at least once. If this were not the case, then at least one agent would never inform other nodes of its initial value: then, since the consensus function is sensitive to all initial values, the algorithm would be unable to correctly compute the consensus function. Transmission of an initial value requires $(\log n + b)$ bytes: its bandwidth complexity is $(\log n + b)/d$.

We can then conclude that the bandwidth complexity of

the consensus problem is $FC(f, \mathcal{G}) \in \Omega((n \log n + b)/d)$. \square

Proposition III.1 strongly relies on the assumption of asynchronous communication. In the next proposition we show that the lower bound on bandwidth complexity also applies to synchronous executions if (i) the pool of UIDs is “large enough” and (ii) the algorithm is required to terminate in a bounded number of steps.

Proposition III.2 (Lower bound on the bandwidth complexity of the consensus problem in synchronous and asynchronous executions). *Assume that messages carry a sender and/or a receiver ID. Assume also that the consensus algorithm terminates within R synchronous rounds and that agents' UIDs are selected from a set S of cardinality $|S| \geq R(n + 1)$. Then, for a given consensus function f and class of graphs \mathcal{G} with n nodes, $FC(f, \mathcal{G}) \in \Omega((n \log n + b)/d)$.*

Proof. In the synchronous setting, if an agent has received no messages from its neighbors, it decides whether to send a message or wait until the next round based on (i) its UID i , (ii) its initial condition x_i and (iii) the number of rounds r elapsed since execution started. For each possible UID i , we call ρ_i the *smallest* round such that, for *some* initial condition $x_{(i, \rho_i)}$, a node with UID i and initial condition $x_{(i, \rho_i)}$ sends a message at round ρ_i if it has not received a message until round $\rho_i - 1$. Informally, ρ_i is the first round when a node with UID i may (for at least one initial condition) send a message if it hasn't received one; $\rho_i - 1$ is the last round when a node with UID i can not send a message unless it has received one, irrespective of its initial condition. The number ρ_i can only assume $R + 1$ possible values (including ∞ if the node never sends a message before round $R + 1$ unless it has received one). Furthermore, at most $n - 1$ UIDs can have $\rho_i = \infty$: otherwise an adversary could build a network where no agent sends a message before round $R + 1$, and therefore consensus is not achieved within R rounds.

Then, by the pigeonhole principle, there exists a round $\bar{\rho}$ with $1 \leq \bar{\rho} \leq R$ such that at least n UIDs have $\rho_i = \bar{\rho}$. An adversary can arrange agents with these UIDs (and matching initial conditions) in a network: then all n agents will send a message at round $\bar{\rho}$, after staying silent for the first $\bar{\rho} - 1$ rounds. Since every message carries the transmitter or the receiver UID, the size of each message is lower-bounded by $\log n$. Thus, $n \log n$ bits may be sent at round $\bar{\rho}$.

Our argument is completed by the observation that, as in Proposition III.1, every agent sends its initial value at least once: this operation has a bandwidth complexity of $\Omega((\log n + b)/d)$.

Thus, the broadcast complexity of the consensus problem in the synchronous case is $FC(f, \mathcal{G}) \in \Omega((n \log n + b)/d)$. \square

We have now proven a lower bound on the bandwidth complexity of the consensus problem for synchronous and asynchronous executions. In the following section we show tightness of this bound by proposing a bandwidth-optimal algorithm, then we compare its performance with commonly-used consensus procedures.

IV. TIGHTNESS OF THE LOWER BOUND AND BANDWIDTH COMPLEXITY OF COMMON CONSENSUS ALGORITHMS

In this section, we analyze the byte complexity of the average-based consensus algorithm [7], [8], [9], the flooding consensus algorithm [6] and the GHS algorithm with convergecast [14], [11]. We also propose a modified version of the GHS algorithm that achieves the lower bound presented in Propositions III.1 and III.2.

A. Bandwidth complexity of common consensus algorithms

Lemma IV.1 (Bandwidth complexity of the average-based consensus algorithm). *Assume that messages carry a sender and/or a receiver UID. Then the bandwidth complexity of the average-based consensus algorithm is $O(n(\log n + b)/d)$, where b is the size of an agent's initial condition.*

Proof. In the average-based consensus algorithm, nodes maintain a local estimate of the consensus value (which has the same type and size as the nodes' initial value). At each time step, nodes send their estimate to their neighbor, then update their estimate as the *average* of their estimate and their neighbors'. Thus, if messages carry the sender UID, each node transmits $\log n + b$ bits with each message. All agents may communicate at once: this is the case, for instance, in synchronous executions, which are a special case of more general asynchronous executions. The resulting bandwidth complexity is $O(n(\log n + b)/d)$. \square

Lemma IV.2 (Bandwidth complexity of the flooding consensus algorithm). *Assume that messages carry a sender and/or a receiver UID. Then the bandwidth complexity of the flooding consensus algorithm is $O(n^2(\log n + b)/d)$, where b is the size of an agent's initial condition.*

Proof. In a flooding algorithm, at each time step, every node sends to all neighbors all *new* information it has received at the previous (asynchronous) round. It is easy to observe that, for certain network topologies (e.g. for the complete graph) every node may receive information from $O(n)$ other nodes at the same time and thus retransmit information from $O(n)$ nodes simultaneously. Each piece of information from one node has size $\log n + b$: thus, all nodes may send messages of size $n(\log n + b)$. Furthermore, all nodes may send large messages simultaneously (e.g., in a synchronous execution). Thus the bandwidth complexity of the flooding algorithm is $O(n^2(\log n + b)/d)$. \square

Lemma IV.3 (Bandwidth complexity of the GHS algorithm with convergecast). *Assume that messages carry a sender and/or a receiver UID. Assume also that the consensus function is hierarchically computable. Then the bandwidth complexity of the GHS algorithm with convergecast is $O((n \log n + nb)/d)$, where b is the size of an agent's initial condition.*

Proof. The GHS algorithm builds a rooted minimum spanning tree by repeatedly merging non-spanning trees across (minimum-weight) edges. The algorithm requires each edge to have a unique weight, while our model considers an

unweighed graph: thus, we assign to each edge a unique, arbitrary, weight¹. Messages exchanged by the agents during execution only contain node IDs, edge weights and boolean values: thus, the size of each message is upper-bounded by $O(\log n)$. Since no message is larger than $\log n$ and nodes only send one (broadcast) message at any given time, the bandwidth complexity of the GHS algorithm is $O(n \log n/d)$. We refer the interested reader to [14] for an in-depth discussion of the algorithm.

Once a rooted tree has been established, the root contacts all other nodes via a tree broadcast and asks them to relay their values. Nodes then relay their values through a tree convergecast: every node waits until it has heard back from all its children (if any), then computes the consensus value for its branch (this is always possible if the consensus function is hierarchically computable) and relays it to its parent. When the root learns every child's consensus value, it computes the overall consensus value and relays it to every node via a tree broadcast.

All messages exchanged in this phase have size $O(\log n + b)$: every message contains a consensus value of size b and a sender and/or a receiver ID. There exist tree network topologies (e.g. a tree of depth one) where up to $n - 1$ nodes may send messages simultaneously during a tree convergecast: thus, the bandwidth complexity of this phase is $O(n(b + \log n)/d)$. This completes our proof. \square

We remark in passing that Awerbuch's minimum spanning tree algorithm [15], which improves the GHS algorithm's time complexity to a time-optimal $O(n)$, is *not* bandwidth-optimal: in particular, the *Test-Distance* procedure has a bandwidth complexity of $O(n \log^2 n)$ during certain executions on selected network topologies, since each Test-Distance message needs to keep track of the return route to its sender.

B. Tightness of the lower bound on bandwidth complexity

The GHS-inspired consensus algorithm outlined above is not bandwidth-optimal: optimality, however, can be achieved with a simple modification that does not influence asymptotic time, message or byte complexity².

The tree-building phase is unchanged. When computing the nodes's consensus function, we exploit the tree structure to make sure that only one node sends a message at any given time. Specifically, once a rooted tree has been established, every node contacts its children *one by one*: children are ordered arbitrarily and every child node is only contacted once the previous child has returned its branch's consensus function. Pseudocode for the algorithm is reported in Algorithm 1.

¹One popular choice for edge weights is to assign to each edge a "weight" equal to the UIDs of the two nodes incident on the edge and use a lexicographic ordering.

²Informally, time complexity is the time required by the algorithm to converge, message complexity is the overall number of messages exchanged by all agents and byte complexity is the overall number of bytes exchanged by all agents. We refer the reader to [11] and [13] for a rigorous definition of these complexity metrics.

Algorithm 1 Bandwidth-optimal consensus function computation on a tree

Input: $IsRoot$ ▷ a bool
 $Children$ ▷ An ordered list of child nodes
 $Parent$ ▷ The parent node's ID
 x ▷ The node's initial condition
 c ▷ A local estimate of the consensus value

if $IsRoot$ is true **then**
 COMPUTECONSENSUSVALUE ▷ The root initializes the consensus procedure
end if

procedure COMPUTECONSENSUSVALUE
 for all $Child$ in $Children$ **do**
 Ask $Child$ to COMPUTECONSENSUSVALUE
 Wait for $Child$'s CONSENSUSREPLY(c_{Child}).
 $c \leftarrow \text{UPDATECONSENSUS}(c, c_{Child})$
 end for
 if $IsRoot$ is true **then** ▷ Inform every node of the consensus value
 Ask $Child$ to RELAYCONSENSUSVALUE(c)
 Wait for $Child$'s ACKCONSENSUS.
 else
 Send $Parent$ a CONSENSUSREPLY(c) ▷ Inform parent of the branch's consensus value
 end if
end procedure

procedure RELAYCONSENSUSVALUE(c_r)
 $c \leftarrow c_r$ ▷ Store the consensus value
 for all $Child$ in $Children$ **do**
 Ask $Child$ to RELAYCONSENSUSVALUE(c)
 Wait for $Child$'s ACKCONSENSUS.
 end for
 if $IsRoot$ is false **then**
 Send $Parent$ an ACKCONSENSUS
 end if
 Terminate
end procedure

function UPDATECONSENSUS(c_i, x_j)
 $x_i \leftarrow c_i * x_j$
end function

It is easy to prove that the bandwidth complexity of this consensus function computation on a tree is $O((\log n + b)/d)$.

Lemma IV.4 (Bandwidth complexity of Algorithm 1). *Assume that messages carry the sender and/or the receiver ID. Assume also that the consensus function is hierarchically computable. Then the bandwidth complexity of Algorithm 1 is $O((\log n + b)/d)$.*

Proof. The algorithm is, essentially, a token-passing algorithm. The token originates at the root. Nodes, starting with the root, pass the token to their children, one at a time, when they ask each child to compute its consensus value; children pass the token back to their parent when they relay the local estimate of the consensus value. Nodes (starting at the root) then pass the token to their children, one at a

time, when they relay the consensus value; children return the token to their parent when they acknowledge reception of the consensus value. It is easy to see that (i) a node only sends a message when it holds the token and (ii) the token is never duplicated, since each node only contacts one child when it has heard back from the previous child and each node only has one parent. Thus, only one node can send a message at any given time. Messages contain a local estimate of the consensus value, a sender ID and a receiver ID: their size is $b + 2 \log n$. The claim follows. \square

Lemma IV.4 allows us to prove tightness of the lower bounds on bandwidth complexity.

Proposition IV.5 (Bandwidth complexity of the consensus problem in asynchronous executions). *Assume that messages carry the sender and/or the receiver ID. Assume also that agents' UIDs are selected from a set S of cardinality $|S| \geq 2n$ and that the consensus function is hierarchically computable. Then, for a given consensus function f and class of graphs \mathcal{G} with n nodes, $FC(f, \mathcal{G}) \in \Theta((n \log n + b)/d)$.*

Proof. In order to prove the claim, we need to find an algorithm with bandwidth complexity $O((n \log n + b)/d)$. The GHS algorithm has a bandwidth complexity of $O(n \log n)$, as shown in the first part of Lemma IV.3. Once a tree structure has been established, Algorithm 1 computes the consensus function with a bandwidth complexity of $O((\log n + b)/d)$. The claim then follows from Proposition III.1. \square

The proof of the next proposition is identical to the one above and is therefore omitted.

Proposition IV.6 (Bandwidth complexity of the consensus problem in synchronous and asynchronous executions). *Assume that messages carry a sender and/or a receiver ID. Assume also that the consensus algorithm terminates within R synchronous rounds and that agents' UIDs are selected from a set S of cardinality $|S| \geq R(n + 1)$. Then, for a given consensus function f and class of graphs \mathcal{G} with n nodes, $FC(f, \mathcal{G}) \in \Theta((n \log n + b)/d)$.*

We note in passing that Algorithm 1 has the same worst-case time, message and byte complexity as the convergecast algorithm proposed in [11], as shown in the two lemmas below.

Lemma IV.7 (Time complexity of Algorithm 1). *The time complexity of Algorithm 1 is $O(n)$.*

Proof. Every node except for the root receives two messages from its parent (one asking to compute its branch's consensus value and one relaying the network's consensus value) and sends two messages to its parent (one informing the parent of the branch's consensus value and one to acknowledge reception of the network's consensus value). Only one message is sent at any given time, as shown in Proposition IV.4. Thus, the time complexity of Algorithm 1 is $4(n - 1)$. \square

Lemma IV.8 (Message and byte complexity of Algorithm 1). *Assume that messages carry the sender and/or the receiver*

ID. Assume also that the consensus function is hierarchically computable. Then the message and byte complexity of Algorithm 1 are $O(n)$ and $O(n(\log n + b))$ respectively.

Proof. As shown in Proposition IV.7, $4(n-1)$ messages are sent in the network. Furthermore, if the consensus function is hierarchically computable, each message has size $(\log n + b)$, as shown in Proposition IV.4. The claim follows. \square

V. A TUNABLE ALGORITHM

The algorithms presented in Section IV offer optimal performance with respect to different complexity metrics.

The flooding algorithm is time-optimal for any graph (and not only worst-case optimal over the class \mathcal{G} of graphs with n nodes); furthermore, it offers *maximal* robustness to disruptions of a communication channel and performs well on time-varying networks. On the other hand, the bandwidth complexity of flooding is the worst among the algorithms we study and its byte complexity (discussed in [11]) is also very suboptimal.

The average-based algorithm performs no better than flooding with respect to byte complexity and robustness, and it has significantly worse time complexity. On the other hand, its bandwidth complexity is better than the flooding algorithm's, but it is not optimal.

The GHS-inspired algorithm has optimal bandwidth and byte complexity. However, the algorithm has minimal robustness margins to the disruption of a communication channel.

In this section we show how the hybrid, tunable algorithm proposed by the authors in [11] achieves *intermediate* bandwidth performance between the flooding algorithm and the GHS-inspired algorithm, recovering the time-optimal behavior of the flooding algorithm and the bandwidth-optimal behavior of GHS for different values of the tuning parameter. This result complements our previous findings on the time and byte complexity of the hybrid algorithm and shows how this algorithm can achieve *mixed* time/bandwidth performance metrics.

Due to space limitations, we only report a high-level description of the algorithm: we refer the interested reader to [11] for details³.

Our algorithm operates in four phases. Phase 1 starts by building a forest of *minimum weight* trees (shown in Figure 3a) of height $O(n/m)$. All nodes run a modified version of the GHS algorithm [14] which only differs from the original algorithm with respect to the stopping criterion. When a node discovers that Phase 1 is over, it informs its neighbors with a broadcast. When a node has finished Phase 1 and all its neighbors have, too, it enters Phase 2.

In Phase 2, tree height is upper-bounded by splitting clusters with too many agents while enforcing a lower bound on tree size. This phase of the algorithm starts at the leaves of each tree. Each node recursively counts the number of its

descendants moving towards the root; agents with more than $\lfloor n/m \rfloor$ offspring create a new cluster, of which they become the root, and cut the connection with their fathers. The tree containing the original root may be left with too few nodes: the root can undo one cut to counter this.

In Phase 3, each tree establishes connections with neighbor clusters, as shown in Figure 3b. When a node switches to Phase 3, it informs all neighbors of its Cluster ID with a broadcast. The root of each cluster is then informed of the available connections to neighbor clusters with a convergecast starting at the leaves. Specifically, as soon as a node knows (i) what clusters its children are connected to (either directly or through their children) and (ii) each neighbor agent's cluster ID, it informs its parent about which clusters it is connected to (either directly or indirectly).

Roots also exploit the tree structure to compute their cluster's consensus function via Algorithm 1: once they have computed the cluster's consensus function, they switch to Phase 4.

In Phase 4, cluster roots communicate with each other through the connections discovered in the previous stage, as shown in Figure 3c. Conceptually, this phase of the algorithm is simply flooding across clusters. Each root sends a message containing its cluster's consensus function to each neighbor tree through the connections built in Phase 3. When a root learns new information, it forwards it once to its neighbor clusters via the same mechanism.

If a link failure breaks one of the trees (as in Figure 3d), the two halves evaluate their size. If either of the two halves is too small, it rejoins an existing cluster; a splitting procedure guarantees that tree height stays bounded. After failure, all nodes update their routing tables.

Finally, when a link outside a tree fails, nodes on the two sides of the failure update their routing tables and notify their cluster roots.

We now study the bandwidth complexity of the tunable consensus algorithm.

Lemma V.1 (Bandwidth complexity of Phase 1 of the tunable algorithm). *Assume that messages carry the sender UID. Assume also that the consensus function is hierarchically computable. Then the bandwidth complexity of Phase 1 of the tunable algorithm is $O(n \log n/d)$.*

Proof. Phase 1 only differs from the GHS algorithm, whose bandwidth complexity is shown to be $O(n \log n/d)$ in Lemma IV.3, with respect to the stopping criterion. When a node stops, it informs its neighbors: the size of the message informing the neighbors is $O(\log n)$, since it only contains the node's UID and a constant-size message. All nodes may stop and inform their neighbors at once: thus the bandwidth complexity of Phase 1 is $O(n \log n/d)$. \square

Lemma V.2 (Bandwidth complexity of Phase 2 of the tunable algorithm). *Assume that messages carry the sender UID. Assume also that the consensus function is hierarchically computable. Then the bandwidth complexity of Phase 2 of the tunable algorithm is $O(n \log n/d)$.*

³We remark two very minor changes with respect to the detailed description in [11]: we replace the challenge-response exchanges at the end of Phase 1 and at the beginning of Phase 3 with broadcasts, to better exploit the *local broadcast* communication model; furthermore, we do not duplicate messages exchanged between clusters for simplicity.

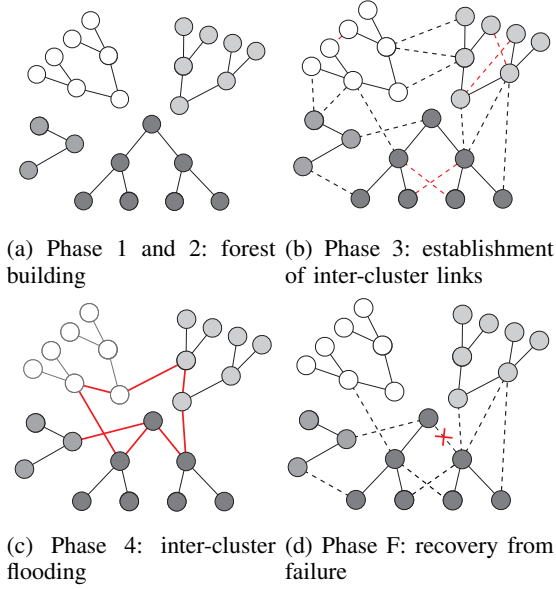


Fig. 3: Schematic representation of the hybrid algorithm behavior.

Proof. The size of all messages exchanged in Phase 2 is $O(\log n)$: each message contains the number of agents that are descendants of the sender node and a cut ID containing two UIDs. Every node has at most $n - 1$ descendants: thus, the number of descendants of a given node can be represented with $O(\log n)$ bits. Since every node sends at most one message at a time, the bandwidth complexity of Phase 2 is $O(n \log n/d)$. \square

Lemma V.3 (Bandwidth complexity of Phase 3 of the tunable algorithm). *Assume that messages carry the sender UID. Assume also that the consensus function is hierarchically computable. Then the bandwidth complexity of Phase 3 of the algorithm is $O((nm \log n + b)/d)$.*

Proof. Two types of messages are exchanged (potentially at the same time) in the cluster discovery routine in Phase 3. First, nodes announce their cluster ID: each message has size $\log n$ (since a cluster ID is the ID of the root node) and every node sends exactly one such message. Then, nodes send their parents a list of the clusters their branch is connected to. Every message contains up to m cluster IDs, each of size $\log n$: thus, the size of each message is upper-bounded by $m \log n$. Every node sends at most one message at any given time: thus the bandwidth complexity of Phase 3 is $O(nm \log n/d)$.

In addition, each root computes the cluster's consensus function with Algorithm 1: the bandwidth complexity of this operation is $O((\log n + b)/d)$, as shown in Lemma IV.4.

Thus, the overall byte complexity of Phase 3 of the algorithm is $O((nm \log n + b)/d)$. \square

Lemma V.4 (Bandwidth complexity of Phase 4 of the tunable algorithm). *Assume that messages carry the sender UID. Assume also that the consensus function is hierarchically*

computable. Then the bandwidth complexity of Phase 3 of the algorithm is $O(nm(b + \log n)/d)$ and $O(m^3(b + \log n)/d)$.

Proof. We prove the two bounds separately. First, we note that every message exchanged in Phase 4 contains a list of at most $m - 1$ intermediate consensus values (one per cluster) and the ID of the relevant cluster: thus, the size of each message is $O(m(\log n + b))$. Since no node sends more than a message at a time, the first bound follows.

Second, we observe that in the cluster flooding algorithm every cluster forwards each new piece of information it receives from a (neighbor or non-neighbor) cluster to neighbor clusters exactly once. Thus, each cluster contacts each of the $O(m - 1)$ neighbor clusters with at most $m(b + \log n)$ bits of information overall.

Each exchange between neighbor clusters may require multiple messages. However, it is easy to see that only one message per exchange is transmitted at a given time: the message is *routed* from the root of a cluster to the root of its neighbor, thanks to the routing tables developed in Phase 3, with no duplication. Thus, even if all m clusters send information about all $m - 1$ other clusters to every neighbor at the same time, no more than $O(m^3(\log n + b)/d)$ bits are exchanged at any given time.

Once a cluster root has computed the overall consensus value, it informs all nodes in its cluster. This is done with Algorithm 1, with bandwidth complexity $O((\log n + b)/d)$. The overall bandwidth complexity of Phase 4 is therefore also upper-bounded by $O(m^3(\log n + b)/d)$. \square

Proposition V.5 (Bandwidth complexity of the tunable algorithm). *The bandwidth complexity of the tunable algorithm is $O(m^3(b + \log n)/d)$ and $O(nm(b + \log n)/d)$.*

Proof. The proof follows immediately from Lemmas V.1, V.2, V.3 and V.4. \square

VI. CONCLUSIONS

In this paper we study the bandwidth complexity of the consensus problem on networks with omnidirectional communication, with particular attention to *robotic* applications. We provide a novel definition of bandwidth complexity which captures the bandwidth use of multi-agent systems with modern Media Access Control mechanisms. This definition allows us to show that, even in network of moderate size, bandwidth use can be a limiting factor on the time performance and on the scalability of common consensus algorithms such as flooding.

We then prove a lower bound on the bandwidth complexity of the consensus problem that becomes tight for hierarchically computable consensus functions and we provide a matching bandwidth-optimal algorithm. Finally, we extend our previous results in [11], proving that the hybrid algorithm presented in the paper achieves intermediate bandwidth complexity between the lower bound and the bandwidth complexity of the time-optimal algorithm, according to a user-defined tuning parameter. The tradeoff between worst-case time performance, byte performance and bandwidth

performance is shown in Figure 1. The implication of this result is that the hybrid algorithm can be used to achieve *mixed* performance metrics, trading time performance and robustness for byte complexity (representative of energy consumption for communication) and bandwidth complexity.

We conclude this paper with a discussion of the limitations of our analysis, which reflect in interesting directions for future research. First, our worst-case analysis provides lower bounds on bandwidth complexity for the class \mathcal{G} of *all* graphs with n nodes: it is of interest to further refine our results to capture the effect of network topology (and in particular of the maximum node degree) on the fundamental limitations on bandwidth performance and on the performance of existing algorithms. Second, an average-case analysis over graphs and asynchronous executions drawn randomly from a representative probability distribution would provide significant insight into the effect of bandwidth complexity on real-world networks, where a (small) probability of collisions is acceptable in presence of a TCP mechanism. Third, while the class of hierarchically computable functions encompasses many relevant engineering problems, it is of interest to study the role of bandwidth complexity for consensus functions that are *not* hierarchically computable. In particular, nonhierarchical algorithms such as average-based consensus may perform no worse than hierarchical algorithms such as GHS with convergecast when the consensus function does not benefit from hierarchical computation. Finally, accurate software and hardware simulations of the performance of the algorithms presented in this paper on wireless channels with different MAC mechanisms will provide further insight into the relevance of our bandwidth metric and into the relative benefits of the different approaches.

REFERENCES

- [1] R. Olfati-Saber, "Distributed Kalman filtering for sensor networks," in *Proc. IEEE Conf. on Decision and Control*, New Orleans, LA, Dec. 2007, pp. 5492–5498.
- [2] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control: Collective group behavior through local interaction," *IEEE Control Systems Magazine*, vol. 27, no. 2, pp. 71–82, 2007.
- [3] Y. Hong, J. Hu, and L. Gao, "Tracking control for multi-agent consensus with an active leader and variable topology," *Automatica*, vol. 42, no. 7, pp. 1177–1182, 2006.
- [4] M. de Weerd and B. Clement, "Introduction to planning in multiagent systems," *Multiagent and Grid Sys.*, vol. 5, no. 4, pp. 345–355, 2009.
- [5] J. N. Tsitsiklis and M. Athans, "On the complexity of decentralized decision making and detection problems," *IEEE Transactions on Automatic Control*, vol. 30, no. 5, pp. 440–446, 1985.
- [6] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.
- [7] J. N. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, Massachusetts Institute of Technology, Nov. 1984, Technical Report LIDS-TH-1424. Available at <http://web.mit.edu/jnt/www/Papers/PhD-84-jnt.pdf>.
- [8] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.
- [9] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003.
- [10] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 33–55, 2009.
- [11] F. Rossi and M. Pavone, "Decentralized decision-making on robotic networks with hybrid performance metrics," in *Communication, Control, and Computing (Allerton)*, 2013 51st Annual Allerton Conference on, Oct 2013, pp. 358–365.
- [12] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.
- [13] F. Rossi and M. Pavone, "On the fundamental limitations of performance for distributed decision-making in robotic networks," in *Decision and Control (CDC)*, 2013 IEEE 52nd Annual Conference on, Dec 2014, in press.
- [14] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 5, no. 1, pp. 66–77, 1983.
- [15] B. Awerbuch, "Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 230–240.